2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Distributed Pursuit-Evasion with Limited-Visibility Sensors Via Frontier-based Exploration

Joseph W. Durham, Antonio Franchi, and Francesco Bullo

Abstract— **This paper addresses a novel visibility-based pursuit-evasion problem in which a team of searchers with limited range sensors must coordinate to clear any evaders from an unknown planar environment. We present a distributed algorithm built around guaranteeing complete coverage of the frontier between cleared and contaminated areas while expanding the cleared area. Our frontier-based algorithm can guarantee detection of evaders in unknown, multiply-connected planar environments which may be non-polygonal. We also detail a method for storing and updating the global frontier between cleared and contaminated areas without building a global map or requiring global localization, which enables our algorithm to be truly distributed. We demonstrate the functionality of the algorithm through Player/Stage simulations.**

I. Introduction

This paper deals with a distributed pursuit-evasion problem for a team of robotic searchers in an unknown environment. The distributed *pursuit-evasion problem*, also known as the *clearing problem*, involves designing control and communication protocols such that the searchers will sweep an environment and detect any intruders which may be present. The pursuit-evasion problem has received a lot of attention in recent years because of its applications to safety and security. In this paper, we describe a distributed environment clearing algorithm based on the concept of the *frontier* or boundary between cleared and contaminated areas. Our algorithm can guarantee the detection of any intruders or, if there are insufficient searchers available, will clear as much area as it can while ensuring no cleared areas are recontaminated.

In the literature on pursuit-evasion problems, many different approaches and starting assumptions have been explored. The study of guaranteeing detection of evaders in planar environments began with [1]. For a single searcher, [2] studied a searcher with a limited field of view in a known polygon while [3] cleared unknown environments without localization using minimalist sensing. Efficient evader detection, where one or more searchers are tasked with probabilistically locating targets which move randomly, is another active area covered in [4]. Pursuit-evasion on graphs representing decompositions of known environments is a related topic which goes back to [5] and includes recent works such as [6] and [7]. In addition, our work draws inspiration from methods for exploration and deployment of agents based on the frontier between explored and unexplored areas, including [8], [9], [10] and [11].

In this paper we present a distributed clearing algorithm for $d-$searchers, a searcher model with realistic limited range visibility sensors. A well-known result from the literature is that computing the minimum number of searchers required to clear a general graph is NP-hard [5]. This result was extended in [12] to searchers with infinite range sensors in a polygonal environment, and so solving for the minimum number of $d-$searchers to clear a non-polygonal environment is also NP-hard. Instead, we present an efficient, distributed algorithm which locally minimizes the number of searchers required, and demonstrate the algorithm's utility through simulations using the opensource Player/Stage robot software system [13].

There are three key contributions of this work. First, our frontier-based algorithm can guarantee detection of evaders in unknown, multiply-connected planar environments which may be non-polygonal. To the best of our knowledge, no prior work exists which achieves this. Second, we detail a novel method for storing and updating the global frontier between cleared and contaminated areas without building a global map or requiring global localization. Finally, we develop a method for selecting the next positions for the searchers which locally optimizes the number of searchers required and the expected increase in the area cleared.

This paper is organized as follows. Section II provides definitions and states the problem which we are addressing. In Section III we examine a centralized version of our algorithm to clarify some of the details. The decentralized algorithm is presented in Section IV and then demonstrated through simulations in V. We conclude with a discussion of future work in Section VI.

II. Problem formulation

We are given a team of n robotic searchers with limited sensing and communication capabilities and finite memory, all initially placed at the same position in the free space of an unknown but limited planar environment. Let Q be the free space of the environment, which must be connected but can have holes and may be non-polygonal. The searchers are tasked with detecting evaders which can be arbitrarily small (even a single point) and can move arbitrarily fast, but continuously, through Q. The trajectories and initial positions of the evaders are unknown. We further require that the

This work is supported in part by NSF awards CMS-0626457 and IIS-0904501 and by ARO MURI award W911NF-05-1-0219.

Joseph W. Durham and Francesco Bullo are with the Department of Mechanical Engineering, University of California, Santa Barbara, CA, 93106 (joey, bullo)@engineering.ucsb.edu

Antonio Franchi is with the Dipartimento di Informatica e Sistemistica, Università di Roma La Sapienza, Italy franchi@dis.uniroma1.it

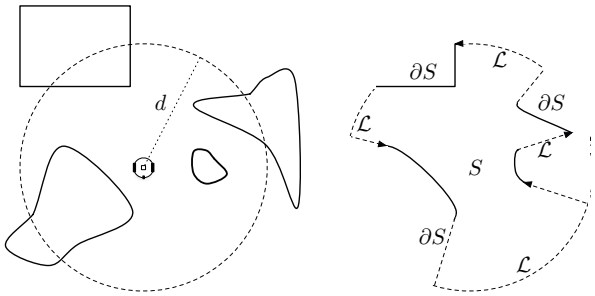

Fig. 1. On the left, four obstacles surround a $d-$searcher and lie within the dashed circular region representing the area perceivable by the searcher's sensor without occlusions. The right image shows the boundary ∂S of the sensor footprint for this position, with dashed oriented arcs for the free boundary $\mathcal{L}$ and solid arcs for the local obstacle boundary.

control protocol for each searcher uses a constant amount of memory with respect to the size of Q.

The robot model we use, the $d-$*searcher*, is a holonomic (i.e., omnidirectional drive) mobile robot that can rotate and translate continuously at bounded speed through Q. Our model gets its name from the attached distance sensor which has a maximum range of $d > 0$ and an angular aperture of 2π. The sensor cannot penetrate obstacles but is capable of detecting any evaders visible to it.

Let S denote the *footprint* of the sensor when a robot is in a generic configuration, as shown in Fig. 1. The footprint is a local obstacle free region and we say that a point is *guarded* by a robot if it belongs to the footprint of the sensor of that robot. The oriented *boundary* of the sensor footprint, ∂S of S, is a closed arc partitioned into two sets: (1) the *local obstacle boundary* (all the points where the sensor has perceived an obstacle), and (2) the *free boundary*, denoted with $\mathcal{L}$, which consists of all the remaining points. Notice that while S is always a simply connected region, $\mathcal{L}$ is not, in general, a connected set. We refer to the connected subsets of $\mathcal{L}$ as *free arcs*. The *orientation* of ∂S is defined in a counter-clockwise manner, such that a point moving along the boundary would have the internal part of S on the left. The free arcs constituting $\mathcal{L}$ inherit the orientation of ∂S and are an open subset of the topological manifold ∂S, with their endpoints on obstacles. The local obstacle boundary arcs, on the other hand, are closed in ∂S.

The *perception* of the sensor at a given point is the tuple $\{S, \partial S, \mathcal{L}\}$, i.e., a footprint S, surrounded by boundary ∂S, and the set of free boundary arcs $\mathcal{L}$ of ∂S. For notation and explanation, we will also have use for the union of a number of perceptions from different points in Q, which we refer to as the *inspected region* I. Since our algorithm does not allow recontamination, I also represents the *cleared area*. We wish to emphasize that our algorithm will not compute or store I, but will instead use only the frontier sections of the boundary of I. Though I will be connected, it may not be simply connected, meaning that ∂I is a set of a closed oriented curves. As with ∂S, ∂I is partitioned into two sets: (1) the *obstacle boundary*, and (2) the set of remaining oriented arcs, called the *frontier boundary*, and denoted $\mathcal{F}$.

Finally, we require that a pair of robots are guaranteed to be able to communicate if their two sensor footprints intersect. We further assume that two communicating robots can compute their relative poses, as a result of a mutual localization procedure [14]. The availability of any sort of global localization is not assumed.

With these definitions we can now state the goal of our algorithm: control the team of n searchers in order to maintain complete coverage of frontier $\mathcal{F}$ while expanding as much as possible the area of the cleared region, I, subject to limited communication and memory constraints.

III. The Multi-robot Clearing Algorithm

For clarity, we have chosen to split the presentation of our clearing algorithm into two stages. In this Section we pretend that a central controller is commanding the searchers in order to describe the fundamental algorithm steps and the data structures involved. In Section IV we detail the distributed implementation of the algorithm.

The team of n searchers is divided into two classes, the *frontier-guards* and the *followers*:

- *Frontier-guard*: Each frontier-guard is assigned to a unique position $v \in Q$ called the guard's *viewpoint*, which can move during the evolution of the algorithm. The frontier-guard must quickly reach its viewpoint and report a perception, i.e. the tuple $\{S, \partial S, \mathcal{L}\}$. In order to detect evaders each frontier-guard must also continuously monitor its sensor.
- *Follower*: Each follower is assigned to follow a frontier-guard, and this assignment can change as the algorithm progresses. Each follower is only required to passively follow its frontier-guard.

As needed, the clearing algorithm will switch frontier-guards to followers, and vice-versa.

At the beginning of the algorithm all n searchers are clustered around a point in Q. One robot is selected as the initial frontier-guard and assigned its initial position as a starting viewpoint. All other robots are set as followers of this guard. The frontier-guard will then record the first perception, which initializes the main data stored during the evolution of the algorithm.

Whenever a frontier-guard records its perception from a viewpoint, a new step k of the algorithm starts and the perception is classified as $\{S_k, \partial S_k, \mathcal{L}_k\}$ and called the k-th perception. We denote the total inspected region at step k as $I_k := \cup_{i=1}^{k} S_i$. Again, the algorithm does not use or store I_k or the obstacle portion of ∂I_k; one important innovation of this work is that it stores and updates only $\mathcal{F}_k$, the oriented frontier arcs of I_k. Since the obstacle boundary of the inspected region I_k is impossible for either searchers or evaders to cross, there are only two ways an evader can enter I_k: (1) by being inside of $S_k \backslash I_{k-1}$ at the instant in which the k-th perception is performed, or (2) by crossing $\mathcal{F}_k$. In this first case detection of the evader is immediate, the focus of our algorithm is thus on maintaining complete coverage of $\mathcal{F}_k$ and updating it when a new perception is added.

After each perception $\{S_k, \partial S_k, \mathcal{L}_k\}$ is recorded, the following actions are performed:

1) Compute $\mathcal{F}_k$ from $\mathcal{F}_{k-1}$ and $\{S_k, \partial S_k, \mathcal{L}_k\}$ as detailed in Sec. III-A.
2) Compute the next set of viewpoints V_{k+1}, which ensure that $\mathcal{F}_k$ remains guarded and that I_{k+1} will be a strict superset of I_k, as detailed in Sec. III-B.
3) Assign each $v \in V_{k+1}$ to a nearby searcher and set the searcher to be a frontier-guard.
4) Assign remaining searchers a frontier-guard to follow.
5) Compute paths for all frontier-guards to reach their assigned viewpoint.

As we will explain in Section III-B, all the points of $\mathcal{F}_k$ will remain guarded by the frontier-guards during the path following. This fact guarantees that at every instant each point of $\mathcal{F}_k$ is guarded by at least one frontier-guard and thus I_k will remain clear. We refer to this feature as the *frontier guarding property*.

Assuming that $n \geq \max\{|V_k| \mid \text{for all } k\}$, the algorithm will terminate at the first step k_f where $\mathcal{F}_{k_f} = \emptyset$. At this point, ∂I_{k_f} will consist entirely of obstacle arcs and I_{k_f} will completely cover Q. Therefore, for every evader e in Q there exists at least one time step k_e during which it (1) crosses $\mathcal{F}_{k_e-1}$ for $k_e \in \{2, \ldots, k_f\}$, or (2) belongs to $S_{k_e} \backslash I_{k_e-1}$ for $k_e \in \{1, \ldots, k_f\}$. We can conclude, by means of the frontier guarding property, that every evader is detected before the algorithm terminates.

A. Updating the global frontier without a global map

On the first step, $\mathcal{F}_1$ is initialized as the free boundary of the first perception $\mathcal{L}_1$; on each subsequent step k, the algorithm needs to compute the new frontier $\mathcal{F}_k$, i.e., the non-obstacle boundary of the inspected region $I_k = I_{k-1} \cup S_k$. The set $\mathcal{F}_k$ can be partitioned into two subsets, (1) the set $\mathcal{F}_{k-1}^{\text{Ext}}$ of arcs from the prior frontier $\mathcal{F}_{k-1}$ which do not belong to the closure of S_k, and (2) the set $\mathcal{L}_k^{\text{Ext}}$ of arcs from $\mathcal{L}_k$ which are not on the interior of $I_k = \cup_{i=1}^{k} S_i$. While the computation of $\mathcal{F}_{k-1}^{\text{Ext}}$ from $\mathcal{F}_{k-1}$ and $\{S_k, \partial S_k, \mathcal{L}_k\}$ is immediate, in this section we describe a new method to compute $\mathcal{L}_k^{\text{Ext}}$ by using only the oriented arcs of $\mathcal{F}_{k-1}$ and $\{S_k, \partial S_k, \mathcal{L}_k\}$.

In all previous work including [11], $\mathcal{L}_k^{\text{Ext}}$ has been computed using S_k and I_{k-1}. The disadvantages of this prior procedure for updating $\mathcal{F}$ are that computing I_{k-1} requires global localization and storing it requires non-constant size per robot with respect to the size of environment Q.

Our new three-step method for computing $\mathcal{L}_k^{\text{Ext}}$ is based around the intersections of the oriented arcs of $\mathcal{F}_{k-1}$ and ∂S_k. As we will discuss in Section IV, this method requires only temporary mutual localization between pairs of agents to compute $\mathcal{F}_k$ and a constant amount of memory per robot to store it regardless of the size of Q. Let $\mathcal{L}_k^\star$ denote the set of points belonging to the intersection between the arcs of $\mathcal{L}_k$ and the arcs of $\mathcal{F}_{k-1}$, and $\bar{\mathcal{L}}_k^\star$ the remaining points of $\mathcal{L}_k$. The points of $\mathcal{L}_k^{\text{Ext}}$ can be either on the boundary of or exterior to I_{k-1}, the boundary points belong to $\mathcal{L}_k^\star$ while the exterior ones belong to $\bar{\mathcal{L}}_k^\star$. The following crucial result states that an arc in $\mathcal{L}_k$ can only switch from being on the interior or exterior of I_{k-1} at an intersection point in $\mathcal{L}_k^\star$.

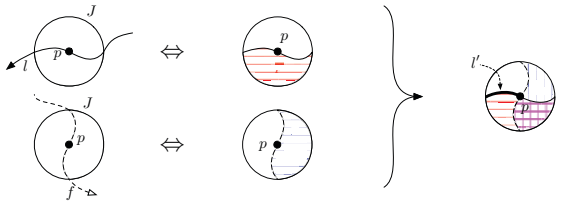

Fig. 2. Example classification of the neighborhood J of a point $p \in \mathcal{L}_k^\star$ where arcs $l \in \mathcal{L}_k$ and $f \in \mathcal{F}_{k-1}$ intersect. In the middle, the partitions of J induced by l and f are represented separately. The white regions on the right side of the oriented arcs indicate the exterior, and the patterned regions on the left indicate the interior. The single neighborhood at right shows the fusion of the two partitions of J. The bold part of l, denoted with l', is classified as belonging to frontier $\mathcal{F}_k$ because it lies between a white and a patterned region. Note that in this case $p \in l'$.

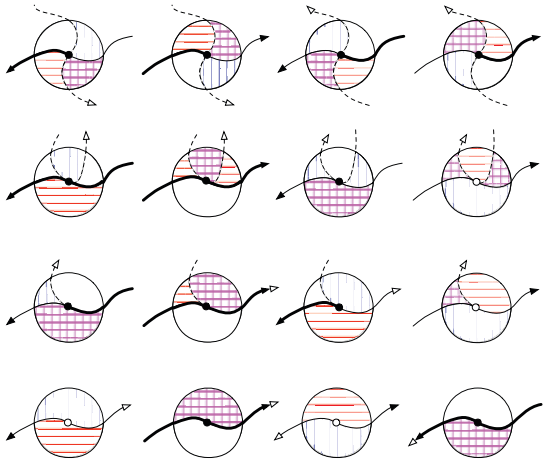

Fig. 3. The classification of the points of arc $l \in \mathcal{L}_k$ in the neighborhood of all possible types of intersections with arc $f \in \mathcal{F}_{k-1}$. Arc l is drawn solid, while f is dashed. Each row shows a different intersection type, with columns for the various reciprocal orientations of l and f. The first row shows isolated crossings, the second shows isolated tangents, the third shows joinings, and the fourth row shows segments where l and f overlap. The bold portions of l are classified as belonging to the new frontier $\mathcal{L}_k^{\text{Ext}}$ as they lie between a white and a patterned region.

Lemma 1. *Let l be an arc in Q which does not intersect $\mathcal{F}_{k-1}$. If any point of l belongs to the exterior of I_{k-1}, then all of l belongs to the exterior of I_{k-1}. If any point of l belongs to the interior of I_{k-1}, then all of l belongs to the interior of I_{k-1}.*

The first step of the method is to classify the neighborhood on ∂S_k of each intersection point $p \in \mathcal{L}_k^\star$ as either internal to I_{k-1} or not. An example of this neighborhood classification is shown in Fig. 2. The neighborhood classifications for all possible intersection cases are depicted in Fig. 3.

The second step of the method is to classify the ends of each arc $l \in \mathcal{L}_k$ in the neighborhood of the endpoints of the adjacent obstacle arcs. These neighborhoods can be classified using the following Lemma:

Lemma 2. *Let o denote a local obstacle arc of ∂S_k, let l_L and $l_R \in \bar{\mathcal{L}}_k^\star$ denote the ends of the free arc segments on the left and right of o, respectively, in the neighborhood of the endpoints of o. Let $E_o \subset o$ be the set of endpoints of any frontier arcs of $\mathcal{F}_{k-1}$ which either begin or end on o, and*

which are, in the neighborhood of o, fully contained in the closure of S_k. Then:

- *If $E_o = \emptyset$, then either l_L and l_R are both internal to I_{k-1} or neither are.*
- *If $E_o \neq \emptyset$, then l_L is internal to I_{k-1} if the closest*[1] *$e \in E_o$ represents the beginning frontier arc, and not internal otherwise. The opposite holds for l_R.*

The third and final step is to propagate the classification from the neighborhoods to all points of the arcs of $\mathcal{L}_k$. This propagation again exploits Lemma 1. Notice that, so long as the selection of viewpoints guarantees that either $L_k^\star \neq \emptyset$ or at least one local obstacle arc o has a non-empty E_o, this third step is well defined.

Combined, these three steps determine which segments of the k-th free boundary $\mathcal{L}_k$ are not in the interior of I_{k-1} and thus should be included in frontier $\mathcal{F}_k$.

B. Viewpoint planner

Our approach to viewpoint planning is similar to the exploration algorithm in [11]. The properties of our viewpoint planning method are laid out in the following Proposition.

Proposition 1. *Given $\mathcal{F}_k$ and the set of prior viewpoints V_k, the viewpoint planner will select the smallest set of viewpoints $V_{k+1} \in I_k$ which satisfy the following constraints:*

1) *Area(I_{k+1}) will be strictly greater than Area(I_k), and*
2) *$\mathcal{F}_k$ is contained in the closure of $\cup_{v \in V_{k+1}} S(v)$.*

Within these constraints, the viewpoint planner will maximize the expected area exposed, Area(I_{k+1}) − Area(I_k).

Remark The viewpoint planner we present here is for circular sensor footprints of radius d. For more general footprints, such as a limited field-of-view, our clearing algorithm could also be applied provided a viewpoint selection method was developed which met the conditions of Proposition 1.

With the distributed application in mind, we simplify the planning of V_{k+1} by constructing it from V_k. Let v_k be the viewpoint of the k-th perception. As detailed in Section III-A, $\mathcal{F}_k$ can be partitioned into two sets: $\mathcal{F}_{k-1}^{\text{Ext}}$ (a subset of the prior frontier), and $\mathcal{L}_k^{\text{Ext}}$ (a subset of ∂S_k). Let $\mathcal{F}_{k-1}^{\text{Int}}$ be the portion of $\mathcal{F}_{k-1}$ which is inside the closure of S_k. The procedure for constructing V_{k+1} is as follows:

1) Remove v_k.
2) Remove any $v \in V_k$ which was assigned to guard an obsolete portion of the frontier in $\mathcal{F}_{k-1}^{\text{Int}}$.
3) Add a set of new viewpoints V' to cover and expand the new frontier segments $\mathcal{L}_k^{\text{Ext}}$.

We will now describe how to choose V' around viewpoint v_k when $\mathcal{L}_k^{\text{Ext}} \neq \emptyset$.

A free arc $l \in \mathcal{L}_k$ is considered *relevant* for viewpoint planning if it contains one or more frontier arc fragments from $\mathcal{L}_k^{\text{Ext}}$. A relevant free arc may contain one or more frontier arc fragments, and each frontier arc fragment will be entirely contained in one relevant free arc. Let $\mathcal{L}_k^{\text{Rel}} \subseteq \mathcal{L}_k$ denote the set of relevant free arcs around v_k.

The goal of this local viewpoint planning can then be restated as partitioning the frontier points of each $l_{\text{Rel}} \in \mathcal{L}_k^{\text{Rel}}$ among the fewest possible new viewpoints V' while maximizing the expected exposed area beyond $\mathcal{L}_k^{\text{Ext}}$.

The first step in our method for local viewpoint planning is to determine how many new viewpoints will be needed to cover each $l_{\text{Rel}} \in \mathcal{L}_k^{\text{Rel}}$. As shown in Fig. 1, each l_{Rel} will be comprised of straight radial segments and circular segments with radius d. The possible configurations are: single radial; single curved; curved with radial on one side; or curved with radial segments on both sides. The following Lemma simplifies the determination of when a radial segment is covered by a viewpoint.

Lemma 3. *Let $v' \in S_k$ be a potential new viewpoint, and $r \in \mathcal{L}_k^{Rel}$ be a radial free arc segment. Let p be the far endpoint of r and $\overline{v'p}$ be the line segment between v' and p. If $\text{dist}(v', p) < d$ and $\overline{v'p}$ only intersects ∂S at p, then open set r will be contained inside of $S(v')$.*

There are two notable consequences of Lemma 3. First, for any l_{Rel} with only a radial segment, one viewpoint is sufficient. Second, for any l_{Rel} which contains both curved and radial segments, we only need to partition the curved segment: the viewpoint which covers an endpoint of the curved segment will also cover any attached radial segment.

To assist in selecting V' we introduce parameter $d_{\min} \in (0, d]$, the minimum distance between v_k and any $v \in V'$. As will become clear, $d_{\min}$ encodes a trade-off in the algorithm: smaller values of $d_{\min}$ reduce $|V'|$ and thereby reduce the number of searchers required; larger values of $d_{\min}$ increase the expected area exposed and thereby reduce the number of iterations required to clear Q.

Let $\delta(l_{\text{Rel}})$ be the angular width of l_{Rel} measured counter-clockwise from the angle of the right-most frontier point on l_{Rel} to the angle of the left-most frontier point on l_{Rel}.

A single new viewpoint at least $d_{\min}$ from v_k can then cover an angular width of at most $\alpha(d_{\min})$ given by

$$\alpha(d_{\min}) = 2\arccos(d_{\min}/2d) \in [2\pi/3, \pi).$$

The number of viewpoints η necessary to cover l_{Rel} is then determined by the following Lemma:

Lemma 4. *For any $l_{Rel} \in \mathcal{L}_k^{Rel}$, η viewpoints will be required where $1 \leq \eta \leq 3$. In particular:*

- *if $\delta(l_{Rel}) \leq \frac{2\pi}{3}$, $\eta = 1$,*
- *if $\frac{2\pi}{3} < \delta(l_{Rel}) < \pi$, $1 \leq \eta \leq 2$,*
- *if $\pi \leq \delta(l_{Rel}) < 2\pi$, $2 \leq \eta \leq 3$, and*
- *if $\delta(l_{Rel}) = 2\pi$, $\eta = 3$.*

For $\eta > 1$, the angular width of l_{Rel} is then partitioned such that the first viewpoint covers $[0, \delta(l_{\text{Rel}})/\eta]$, and each subsequent viewpoint covers the next equally sized slice.

Once we know how many v' are needed to cover each l_{Rel}, the final step is to optimize the placement of each v' to maximize the expected area it will expose. For every v' there are two points, p_1 and $p_2 \in l_{\text{Rel}}$ which must be covered:

[1] With respect to the distance on the arc o.

for a single radial segement, p_1 and p_2 are the endpoints of the segment; for any other configuration, p_1 and p_2 are the endpoints of the partition of the curved arc in l_{Rel} assigned to v'. Let $S_k(p_1)$ and $S_k(p_2)$ be the subsets of S_k which are known to be visible from p_1 and p_2, respectively. We then choose v' as a point in $S_k(p_1) \cap S_k(p_2)$ which minimizes the sum of the distances to each frontier point in the partition of l_{Rel} assigned to v'.

By construction, this method of selecting V' guarantees that $\mathcal{L}_k^{\text{Ext}} \in \cup_{v' \in V'} S(v')$. The following Lemma states that it also ensures that $\text{Area}\,(I_{k+1}) - \text{Area}\,(I_k) > 0$:

Lemma 5. *For each $v' \in V'$, $S(v')$ will cover some new area $A \in Q$ where $\text{Area}\,(A) > 0$ and $\text{Area}\,(A \cap I_k) = 0$.*

IV. The Distributed Clearing Algorithm

For the distributed clearing algorithm the communication graph is in general disconnected, necessitating a couple changes from the centralized description. First, the global frontier must be stored and updated in a distributed manner. Second, viewpoint planning must be performed locally by the frontier-guards. Furthermore, the distributed algorithm cannot rely on a global localization system. Finally, while the centralized version is synchronous, in the distributed setting it is possible for perceptions from disconnected searchers to be recorded at the same time.

We distribute the global frontier by having each frontier-guard store its local frontier segments and update them through communication with its neighboring frontier-guards. This distributed storage and updating can always be achieved, since (1) by the frontier guarding property, each global frontier point is guarded by a frontier-guard; (2) the classification of $\mathcal{L}_k$ requires only those frontier segments which intersect it, and by assumption two robots whose footprints intersect are in communication and are mutually localized (e.g., by the method described in [14], or by scan matching).

Once the local frontier for a frontier guard has been classified, viewpoint planning relies only on the local information. In addition, the execution of the path between viewpoints can be done without global localization; since both viewpoints lie inside the local perception, either local odometry of reasonable accuracy or a registration of the footprints taken from the two viewpoints will suffice. In fact, frontier updating is also based only on current relative positions, not the absolute position. The distributed algorithm can, therefore, continue to clear an environment even if the searchers cannot determine where they started.

The two classes of searchers from the centralized algorithm are each split in two, yielding four possible states:

- *Expand*: When a searcher is assigned a new viewpoint to move to, it enters the expand state until it reaches the viewpoint and records a perception.
- *Frontier-guard*: Each frontier-guard i will remain stationary at its viewpoint and has complete control over its local frontier segments, $\mathcal{F}_{k,i}$. It must communicate with neighboring frontier-guards to keep $\mathcal{F}_{k,i}$ updated, plan new viewpoints to cover and expand $\mathcal{F}_{k,i}$, and then dispatch a followers to the new viewpoints.
- *Follow*: Followers passively follow and respond to commands from their frontier-guard.
- *Wander*: Wanderers are followers who have not found a frontier guard to follow. When a frontier-guard has no more local frontier to guard, it and its former followers must wander until they locate a frontier-guard to follow.

The four-state state machine and the distributed algorithm are explained in Fig. 4 and 5. The key subroutines are:

- `UpdateNeighbFrontier/Frontier`: These two functions perform a localized version of the frontier update method in Section III-A. Searcher i first acquires its neighbors' current frontier segments, classifies $\mathcal{L}_{k,i}$ using these segments, and then informs its neighbors if any of their frontier segments lie within $S_{k,i}$.
- `ViewPointPlan`: This function follows the viewpoint planning method in Section III-B. It determines how many viewpoints are needed based on the number and angular width of the relevant free arcs. Then, the single best new viewpoint is chosen from $S_{k,i}$.
- `PathToViewPoint`: Determines the shortest path from the old viewpoint to the new viewpoint inside $S_{k,i}$, which will be a straight line if $S_{k,i}$ is star-shaped.
- `SearchForLeader`: This function does a random walk with two additional behaviors. If a wanderer encounters a frontier-guard or expander, then it switches to following this leader. If two wanderers come in contact, they may join together to form a wandering blob.

The behavior of the frontier-guards in this distributed clearing algorithm guarantees the frontier guarding property from Section III. When expander i reaches its viewpoint and makes a perception, it then enters the stationary frontier-guard state. So long as i remains a frontier-guard, it will maintain complete coverage of the frontier segments in $\mathcal{F}_{k,i}$. Searcher i will only leave the frontier-guard state if either $\mathcal{F}_{k,i}$ is erased by a new neighbor, or if i determines that one new viewpoint is sufficient to cover $\mathcal{F}_{k,i}$ and that the path to the viewpoint also maintains coverage of $\mathcal{F}_{k,i}$.

The combination of the frontier guarding property and Lemma 5 guarantees that the distributed algorithm will successfully clear all of Q, assuming there are sufficient searchers available. When the task is completed, all searchers will be in the Wander state. If all-to-one communication is available (i.e., if all robots can communicate back to a central security center), then detecting task completion is trivial. In the most general case, the searchers will have to determine the task is complete by querying the other searchers. In the absence of global localization or other means of assuring rendezvous, our proposal is that robots in the Wander state clump together when they encounter each other to form wandering blobs. Eventually, through the random walks of these growing blobs, all searchers will be joined into a single blob and task completion can be easily detected.

V. Simulations

To demonstrate the utility of the proposed distributed clearing algorithm, we implemented it in the open-source Player/Stage robot software system [13] using the Multirobot

Procedure `Expand`

```
Data: frontier,path
foreach follower in followers do
    Send(follower,"follow",path);
Move(path);
{S, ∂S, L} ← Perceive();
neighbFront ← UpdateNeighbFront();
frontier ← Frontier({S, ∂S, L},frontier,neighbFront);
DoBehavior("Frontier-Guard",S,frontier);
```

Procedure `Frontier-Guard`

```
Data: S,frontier
if frontier is empty then
    Send(followers,"wander");
    DoBehavior("Wander");
(bestVP,NumVPs) ← ViewPointPlan(S,frontier);
path ← PathToViewPoint(S,bestVP);
if NumVPs == 1 then
    DoBehavior("Expand",frontier,path);
else
    if followers has at least one follower then
        follower ← PopFollower(followers);
        Send(follower,"expand",path);
        WaitForFollower(follower);
    else
        while no new neighbor and no followers do
            Sleep();
    DoBehavior("Frontier-Guard",S,frontier);
```

Procedure `Follow`

```
Receive(Leader,message,path);
switch message do
    case "follow"
        Move(path);
    case "expand"
        DoBehavior("Expand",∅,path);
    case "wander"
        DoBehavior("Wander");
```

Procedure `Wander`

```
SearchForLeader();
if leader found then
    DoBehavior("Follow");
if all searchers wandering then
    exit
```

Fig. 4. Details of the actions taken by searchers in each of the four possible searcher roles: Expand, Frontier-guard, Follow, and Wander.

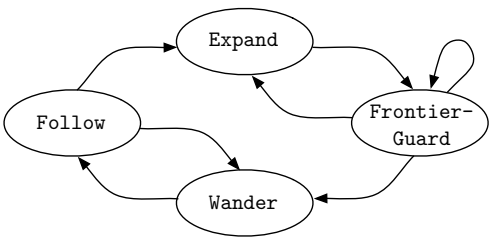

Fig. 5. State machine diagram for the distributed clearing algorithm.

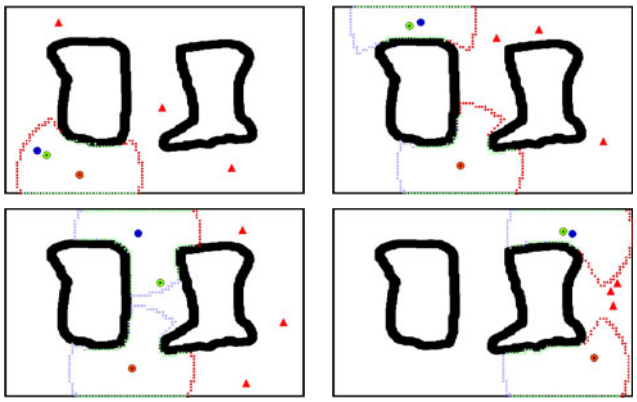

Fig. 6. A simulation of three circular robots sweeping an environment with holes to locate the triangular evaders, with images ordered left-to-right and then top-to-bottom. The discretized boundary of each frontier-guard's current footprint, ∂S, is shown using colored squares where dark-red is used for local frontier segments.

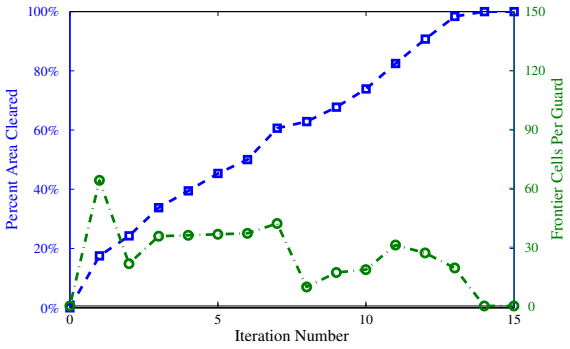

Fig. 7. For the simulation in Fig. 6, the percentage of the total free space cleared at each iteration is plotted with blue squares against the left axis, while the average frontier cells stored per frontier-guard is plotted with green circles against the right axis.

Integrated Platform [15]. Perceptions are implemented as local occupancy grids, with oriented frontier arcs handled as ordered sequences of cells. Each robot stores only its most recent perception and its local frontier.

Three simulations of the algorithm in action are shown in the video included with the submission of this paper and are described in this Section. The first simulation features three searchers clearing an environment with two large holes and three evaders. The progression in Fig. 6 begins at the top-left, where the robots start expanding from a corner of the map. The second screenshot shows the searcher clearing the lower passage waiting for help to cover one of its two frontier segments. Once the central vertical passage is cleared by another searcher, the trio continue on to complete their sweep. Fig. 7 shows the total percentage of the free space cleared by the robots for each iteration (where a new iteration begins with each recorded perception), as well as the number of frontier cells stored per frontier-guard.

In the second simulation, six searchers track down five evaders in a larger multiply-connected environment. Two screenshots are shown in Fig. 8, where the robots begin in

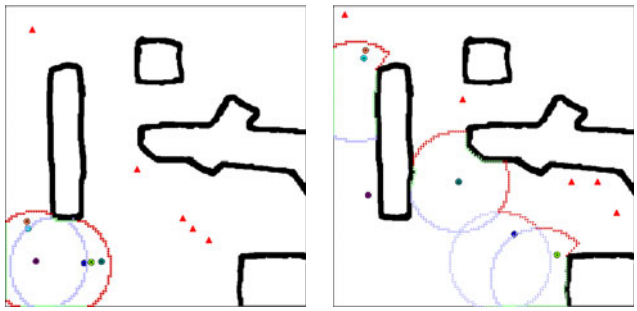

Fig. 8. Two screenshots from a simulation of six circular robots clearing an environment containing five triangular evaders.

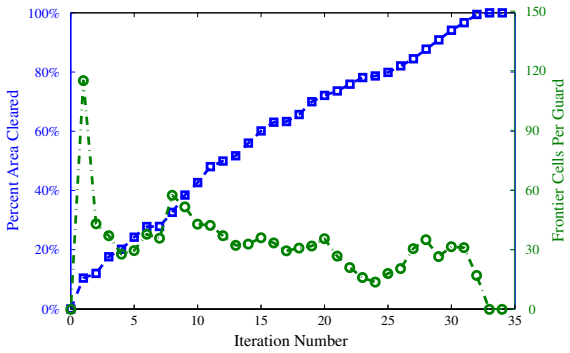

Fig. 9. For the simulation in Fig. 8, the percentage of the total free space cleared at each iteration is plotted with blue squares against the left axis, while the average frontier cells stored per frontier-guard is plotted with green circles against the right axis.

a corner of the map before spreading out to cover the free space. Notice that in the second image there are two groups of searchers which are separated and cannot communicate with each other. Fig. 9 shows the total area cleared and the frontier cells stored over the iterations of the algorithm. This plot shows that the number of frontier cells stored per frontier-guard is independent of the area cleared.

The third and final simulation consists of twelve searchers expanding in a vast empty environment. Using only the local viewpoint optimizations discussed in Section III-B, the paths taken and the final configuration shown in Fig. 10 closely approximate the globally optimal trajectories where the searchers fan out until they are equally spaced on the circumference of a large circle with their sensor footprints just touching.

VI. Future Work

There are a number of interesting future directions for this work. First, the specification of a general viewpoint planner for sensors with a limited field-of-view would extend the algorithm to a much broader class of hardware. The development of bounds on the number of $d-$searchers required to clear an environment would also be a significant contribution. Another extension would be to guarantee a connected communication graph for the searchers at all times, perhaps including a connection back to the initial starting point.

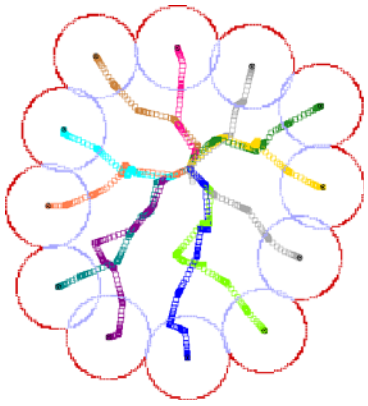

Fig. 10. Final configuration of 12 robots clearing as much area as the can in an empty environment, with tracks showing the paths of each robot.

Finally, the frontier concept could also be applied to three-dimensional environments.

References

[1] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," *SIAM Journal on Computing*, vol. 21, no. 2, pp. 863–888, 1992.

[2] B. P. Gerkey, S. Thrun, and G. Gordon, "Visibility-based pursuit-evasion with limited field of view," *International Journal of Robotics Research*, vol. 25, no. 4, pp. 299–315, 2006.

[3] S. Sachs, S. Rajko, and S. M. LaValle, "Visibility-based pursuit-evasion in an unknown planar environment," *International Journal of Robotics Research*, vol. 23, no. 1, pp. 3–26, 2004.

[4] G. Hollinger, S. Singh, J. Djugash, and A. Kehagias, "Efficient multi-robot search for a moving target," *International Journal of Robotics Research*, vol. 28, no. 2, pp. 201–219, 2009.

[5] T. D. Parsons, "Pursuit-evasion in a graph," in *Theory and Applications of Graphs*, ser. Lecture Notes in Mathematics, Y. Alavi and D. Lick, Eds. Springer, 1978, vol. 642, pp. 426–441.

[6] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B.Vöcking, "Randomized pursuit-evasion in graphs," *Combinatorics, Probability and Computing*, vol. 12, no. 3, pp. 225–244, 2003.

[7] A. Kolling and S. Carpin, "Multi-robot surveillance: an improved algorithm for the graph-clear problem," in *2008 IEEE Int. Conf. on Robotics and Automation*, Pasadena, CA, May 2008, pp. 2360–2365.

[8] B. Yamauchi, "Frontier-based exploration using multiple robots," in *2nd Int. Conf. on Autonomous Agents*, Minneapolis, MN, May 1998, pp. 47–53.

[9] A. Howard, M. J. Matarić, and G. S. Sukhatme, "An incremental self-deployment algorithm for mobile sensor networks," *Autonomous Robots*, vol. 13, no. 2, pp. 113–126, 2002.

[10] A. Ganguli, J. Cortes, and F. Bullo, "Distributed deployment of asynchronous guards in art galleries," in *2006 American Control Conference*, Minneapolis, MN, Jun. 2006, pp. 1416–1421.

[11] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli, "The sensor-based random graph method for cooperative robot exploration," *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 2, pp. 163–175, 2009.

[12] L. J. Guibas, J.-C. Latombe, S. M. Lavalle, D. Lin, and R. Motwani, "A visibility-based pursuit-evasion problem," *International Journal of Computational Geometry & Applications*, vol. 9, no. 4/5, pp. 471–494, 1999.

[13] B. Gerkey and contributors, "The Player/Stage Project," `http://playerstage.sourceforge.net`, July 2009, version 2.13.

[14] A. Franchi, G. Oriolo, and P. Stegagno, "Mutual localization in a multi-robot system with anonymous relative position measures," in *2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, St. Louis, MO, Oct. 2009, pp. 3974–3980.

[15] A. Franchi and P. Stegagno, "Multirobot Integrated Platform," `http://www.dis.uniroma1.it/ labrob/software/MIP/`, Aug. 2009.